Story Writing Tips

Last edited by **Wayne Starr** 4 weeks ago

Below are some tips to keep in mind while doing story writing. Note this is not all-encompassing, and story writing is definitely more of an art than an exact science. If you would like to practice your story writing, you can take a look at the Lunch and Learn that was done on Story Writing, linked below:

https://gitlab.platform1.ninja/levelup/lunch-and-learns/story-writing-workshop/-/tree/master

1. Focus on outcomes, not checklists or todos

The first tip (and possibly one of the more important tips) is to keep stories (whether they be chores, bugs, or features) focused on the outcomes desired from the task versus being prescriptive on the exact tasks to accomplish within a story. This allows the developers who pickup the story to decide what architecture is best while working through issues, and affords them more flexibility. Below is an example of a bad and a good story from this perspective.

NOT RECOMMENDED

```
Create a view profile page within the application.

[ ] Add a button to the home page to navigate to the profile
[ ] Create a profile page
[ ] Add the user's name, email, and phone number to the page

**Notes:**
The user interface should follow the attached design
This feature is for viewing the profile only (no editing)
```

RECOMMENDED

```
As Robert I want to be able to view my profile so that I can see what information the

**Given** I have logged into the application
**And** am on the home screen
**When** I click a new user profile icon in the upper-right corner
**Then** I will be taken to a "My Profile" page
**And** I will see my name, email, and phone number listed on screen

**Notes:**
The user interface should follow the attached design
This feature is for viewing the profile only (no editing)
```

As you can see, both stories contain roughly the same information, but focusing on outcomes allows the developers to decide what steps are best to implement the story as well as giving them context on how the user interacts with the application.

2. Use Gherkin Syntax for Acceptance Criteria

Gherkin Sytntax is a way to describe acceptance criteria that allows you to more immediately write tests by simply following the flow of the criteria. It is also useful to hold yourself to writing stories toward outcomes since it walks through what a user would expect as they walk through the system instead of being a prescriptive list of things to do. Below is an example of Gherkin for a normal application user story:

```
Given I am logged into the application
And am on the home screen
When I click a new user profile icon in the upper-right corner
```

```
Then I will be taken to a "My Profile" page
And I will see my name, email, and phone number listed on screen
```

As you can see in this example, Gherkin allows you to be very prescriptive without getting into the weeds on how any of that is done. It also allows you to add a lot more detail than may otherwise be in regular acceptance criteria. In general, below are what the main pieces of the syntax are used for:

- **Given**: This is the state of your *application* that needs to be in place before the user story starts. Because this is for the application/software, it should focus on the exact place in the flow the user is in.
- **When**: This is the trigger/action that is performed for your application to do something. This can either be from a human or a machine, but is usually from the persona the story has been written around.
- **Then**: This describes the response that your application or software will give to the trigger/action. This should be specific again to the state, describing exactly how the application responds.
- **And**: This is used to chain multiple statements of a Given, When or Then together. Note that if you have too many of these, you may need to break your story down into atomic. chunks (discussed below).

Gherkin even can be applicable in cases where you aren't really talking about a user interface at all (since it is more about user flow), and can even help platform developers document interactions between services as in the example below.

```
Given I make an authenticated request through GCDS
When the request enters the cluster
Then it will be routed through "louketo-proxy"
And "louketo-proxy" will handle translating the OIDC attributes to a JWT
And the JWT will be passed along in the "Authorization" header to the application
```

Users also don't have to be humans, as will be described in the next section, and could simply be an external system that is interacting with your application in some way (such as a sensor picking up data down range).

3. Make use of personas to better define stories

Personas allow you to add more context to a user story without needing a lot of background and boilerplate within the story itself. They describe a given user of the system (whether human or not), and provide some attributes of them that are both relevant, as well as relatable. While they may seem cliche, they allow the developers to have more information about the users of the system which can help them point out inconsistencies when implementing stories instead of simply following along with exactly what the story prescribes. Ultimately this helps to build better software, and makes for happier developers as they are able to empathize with and better understand their user base. Below is an example of a user persona:

```
**Robert**
Robert, 26, is looking to become an owner-operator of his own independent trucking
business that employs only himself.  He just purchased a single truck and trailer,
and interacts with the MCS-150 form only as necessary.  He doesn't need to compile
a ton of data to fill in the form, and for the future will largely use the same
data year over year.  He is not very technical, but does what he can to understand
the basics.
```

4. Break stories down into the smallest possible chunks

Finally, you should break stories up into atomic chunks that cannot be split up further. This ensures that stories are more easily understood by developers, and also allows them to be picked up and put down quickly. The latter is important because context-switching can impede progress, and the longer a story is up (especially if it remains up over a weekend), then the more context is lost, and the more rework that needs to be done. It is also useful in planning to have shorter stories since you can more quickly pivot from one story to the next, and can bucket the team's time much more efficiently.